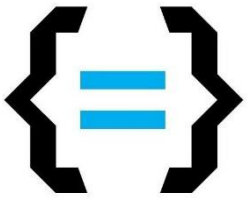


Fakulta informatiky a managementu

Univerzita Hradec Králové

Rozpoznávání objektů v obraze

Vypracoval: Pavel Kratochvíl

Studijní obor: Aplikovaná informatika

Datum vypracování: 13.8.2017

Vedoucí práce: Ing. Ondřej Klapka

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a uvedl veškeré literární prameny, které byly během této práce použity. Zároveň souhlasím se zveřejnění této práce jak v tištěné, tak v elektronické podobě.

V Hradci Králové dne 16. 8. 2017 _____

Pavel Kratochvíl

Poděkování

Rád bych poděkoval Ing. Ondřeji Klapkovi za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce.

Abstrakt

Cílem této bakalářské práce je popsat možnosti předzpracování obrazu a jeho následného zpracování v algoritmu neuronových sítí s cílem rozpoznávání objektů. Dále popsat princip algoritmu neuronových sítí a navrhnout implementaci aplikace pro rozpoznávání číslic, která bude založena právě na neuronových sítích. V poslední části bude rozebrána praktická aplikace pro rozpoznávání číslic.

Abstract

The main goal of this bachelor's thesis is to describe options for image pre-processing and the following recognition process of the information in picture. Additional goal is to describe the algorithm of neural network and implement a working solution for number recognition, which will be based on this algorithm. In the last section, the implementation itself will be covered there.

Obsah

1 Úvod

Rozpoznávání objektů v obraze je v současné době automatizace mnoha oblastí lidské činnosti velmi populární a skloňované téma. Rozpoznávání slouží k identifikaci objektů reálného světa na fotografiích nebo videích. Toto téma je úzce spjato s tématem počítačového vidění a zpracování obrazu. Je založeno na myšlence, že každý objekt má své speciální vlastnosti, na jejichž základě je můžeme identifikovat. Jako příklad si můžeme představit lidský obličej, kde vlastností může být vzdálenost mezi očima, barva očí, oči samotné nebo nos. Rozpoznávání objektů je hodně využíváno například v biomedicíně, potravinářském průmyslu, elektronickém průmyslu nebo strojírenství. Škála využití strojového učení je velice široká. Konkrétními příklady použití je rozpoznáváni tvarů nakreslených na dotykovém displeji nebo myší. Zde je vhodné zmínit projekt Quickdraw od společnosti Google, která se v něm chlubí svou neuronovou sítí na rozpoznávání nakreslených objektů. Dalším příkladem jsou programy pro zpracování rentgenových snímků pacientů s podezřením na rakovinu. Všudypřítomná zobrazující se reklama na webových stránkách, tedy konkrétně její obsah, je vybírána také neuronovou sítí, která je schopna do jisté míry předpovídat, co člověk zrovna potřebuje a po čem prahne. V neposlední řadě by mohly být zmíněny spam filtry v e-mailových schránkách, bez kterých by tyto schránky byly zahlceny nechtěnou, a velice často i nebezpečnou poštou. [4]

Počátky výzkumu počítačového vidění datujeme do brzkých 60. let 20. století. První rozpoznávací systémy sloužily pro rozpoznávání písmen pro kancelářské využití. V těchto letech patřila mezi nejvýraznější společnosti zabírající se tímto tématem firma Hitachi Labs v Japonsku, odkud pochází výraz machine vision neboli strojové vidění. Strojové vidění je pragmatičtější odnož počítačového vidění, která si dávala za úkol nahradit lidské dělníky. První plně automatizovaný stroj využívající tuto myšlenku byl nasazen v roce 1973 a sloužil pro verifikační proces při výrobě polovodičů. [1]

Proces rozpoznávání obrazu je vhodné rozložit na 2 podprocesy a každý řešit samostatně. Pro odstranění nadbytečných informací a dat z obrazu slouží proces předzpracování. Výstupní data tohoto subprocesu jsou následně použita jako data vstupní pro proces druhý, kterým je samotné rozpoznávání objektů.

Cílem této práce bude prozkoumat algoritmus neuronových sítí a jemu předcházející kroky úprav pro dosažení přesných výsledků. Dále prozkoumanou problematiku převést do algoritmů a za pomoci vybraného programovacího jazyka naimplementovat.

2 Předzpracování obrazu a segmentace

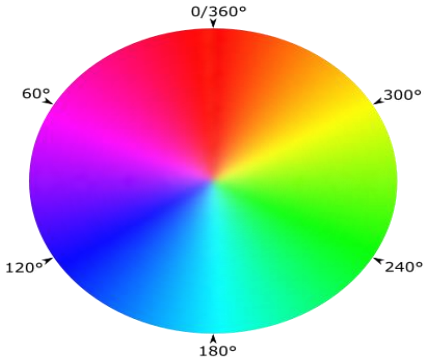
Snímek je po svém pořízení často nějakým způsobem poškozen. Příkladem poškození může být šum nebo zkreslení. Fáze předzpracování se snaží tyto nedostatky v obraze zredukovat, případně úplně potlačit, a za cíl si nedává nic menšího než zvýšení kvality obrazu pro další zpracování. Segmentace je pak navazující proces, ve kterém se obraz rozdělí na několik menších, z nichž každý by měl obsahovat právě jednu souvislou část, tedy jeden objekt. A na tomto objektu bude následně možné spustit algoritmus rozpoznávání.

2.1 Převod obrazu do odstínů šedé

Pro práci s obrazem ve stupních šedi je mnoho důvodů. V první řadě je na barvu možné hledět jako na šum nebo rozptýlení, které ve většině případů rozpoznávání objektů ničemu nepomůže a pouze zvětšuje výpočetní náročnost celého procesu. Mimo jiné algoritmy předzpracování, které mohou být na obraze v odstínech šedi jednoduché s barevnou hloubkou, nabývají složitosti. V neposlední řadě získáme redukcí barevné hloubky rychlejšího dosažení požadovaných výsledků, neboť každý pixel v modelu RGB (24 bitů) nabývá hodnot od 0 do 255 na třech barevných kanálech, potažmo čtyřech v modelu RGBA. Po převedení do odstínů šedi se počet kanálů sníží na jeden. Barevná hloubka se zredukuje na 8 bitů. To odpovídá hodnotám od 0 do 255 na jednom kanále, tedy přípustných 256 hodnot, což v detekci objektů výrazně usnadní výpočetní složitost, protože nadále již nebudeme muset zpracovávat třírozměrné pole.

Existují samozřejmě i výjimky, kdy by měla být barevná hloubka zachována, protože bez ní by byl rozdíl mezi objekty ztracen – pokud je v obraze například zachycen oranžový, potažmo nažloutlý, objekt před světle zelenou. Obecně lze říci, že barva je důležitý faktor, pokud barva objektu svírá s barvou pozadí malý úhel na

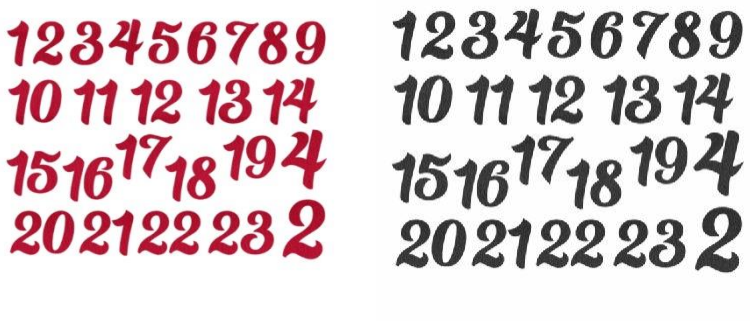
HSV, potažmo HSL kružnici. To se ale ve většině případů, v kterých chceme použít detekci objektů, nestává. Obrázek: 1 HSV kružnice



Obrázek: 1 HSV kružnice

Implementovat převod obrazu do odstínů šedi lze více způsoby. Prakticky jde vždy o průměrování hodnot barev pixelu. Nejjednodušší možností, jak obraz barevný transformovat do odstínů šedi, je využití metody lightness. Ta na každém pixelu zprůměruje nejvyšší z hodnot R, G, B s tou nejmenší, je tedy dána vzorcem: [2]

$$f(x) = \big(max(R,G,B) + min(R,G,B)\big)/2.$$



Obrázek: 2 Převod do odstínů šedi s využitím metody luminosity

Další možností je průměrování všech tří barevných kanálů. Vzorec odpovídá předpisu *f(x)=(R+G+B)/3*, kde *x* je výsledná hodnota pixelu a R, G, B jsou zástupné symboly pro červený, zelený a modrý kanál. Výčet možností pro dosažení stejné transformace, leč s lehce odlišným výsledkem, je završen metodou luminosity – nejsofistikovanější ze všech možností převodu obrazu do odstínů šedi. Stejně tak jako předchozí metoda se hodnota barevných kanálů průměruje, ale v tomto případě jde o průměr vážený. Váha každé barvy je úzce spjatá s vnímáním barev lidským okem. Člověk je schopen nejvíce vnímat zelenou barvu. Tomuto kanálu je tedy přiřazena

největší váha. Pro každý pixel je aplikována funkce s využitím stejným zástupných znaků jako u metody průměrování. [2]

$$f(x) = 0.21 * R + 0.72 * G + 0.07 * B$$

2.2 Odstranění šumu

Ať je již obraz barevný transformován na obraz černobílý, nebo je v původních barvách, bývá poškozen chybami. Chyba může vznikat v různých částech pořízení snímku. Ať už je to při pořízení například nedostatečným nasvícením, nebo při převodu jednoho obrázkového formátu do jiného například kvůli kompresi. Každý nesyntetický obraz tedy obsahuje kromě skutečných informací o obsahu ještě nenulovou míru šumu.

Tato míra šumu může v dalších krocích zpracování obrazu zapříčinit chybovost, neboť se lokálně chová jako hrana. Hranové detektory jsou založeny na skokové změně jasové funkce, kterou způsobuje i již zmíněný šum. Kvůli tomuto chování je tedy potřebné šum odstranit. Toho je možné dosáhnout filtrací pomocí Gaussova filtru, popřípadě průměrováním hodnot pixelů s hodnotami v jeho okolí. Pro tyto účely je využíváno postupu zvaného konvoluce. To je vlastně aplikace transformační funkce, pomocí které získáme ze vstupních hodnot hodnoty jiné. Tento vztah je zachycen následující rovnicí: [3, 10]

$$g(x,y) = f(x,y) * h(x,y) = \sum_{i=-i}^{k} \sum_{j=-j}^{k} f(x-i, y-j) * h(i,j).$$

V rovnici je na původní hodnotu danou vztahem $f(x,y)$ aplikována maska *h(i,j)* V případě průměrování je maska *h* obvykle čtvercová matice s lichým počtem prvků v řádce a kde každý prvek matice má hodnotu $1/N$, přičemž N je celkový počet prvků v matici. Funkce tedy prochází celý obraz a každý pixel a hodnoty jeho okolí vynásobí hodnotami z konvolučního jádra. Výsledné hodnoty jsou následně sečteny a hodnota tohoto součtu je přiřazena právě zpracovávanému pixelu na souřadnicích x, y. Konvoluční maska využívaná při průměrování je tedy zapsána jako:

$$h(i,j) = 1/9 * \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

Alternativou může být využití Gaussova filtru. Tato metoda také využívá konvolučního jádra. Nicméně hodnoty jednotlivých prvků konvolučního jádra nejsou pouze jednoduchým průměrem, ale průměrem váženým. Váha prvku je dána exponenciální vzdáleností od zpracovávaného pixelu x. Konvoluční jádro h(i,j) je tedy čtvercová matice, se kterou je provedena původní obrazové funkce f(i,j) stejně jako u metody předchozí. Gaussova konvoluční maska je dána maticí: [3]

$$\begin{matrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{matrix},$$

což je v podstatě vážený průměr, kde váhou je vzdálenost od středu matice.

2.3 Detekce hran

Po provedení operace redukce šumu lze přistoupit k detekování hran objektů nacházející se v obraze, neboť byl odstraněn matoucí šum, který by algoritmus detekce hran mohl nesprávně vyhodnotit jako hranu. K hranové detekci se nejčastěji využívají detektory, které analyzují jasovou funkci. V místě, kde se nachází hrana, dochází ke skokové změně této funkce. Tyto změny je možné nalézt výpočtem první derivace. Hrana se bude nacházet v maximech tohoto výpočtu. Další, přesnější možností je hledání průchodů nulou po výpočtu druhé derivace, nicméně ne tak hojně využívaná kvůli výpočetní náročnosti. Pro detekci hran mohou být použity detektory Robertsův, Prewittové, Sobelův, Robinsonův, Kirshův, Laplaceův v čtyřokolí a Laplaceův v osmiokolí. Tyto hranové detektory aproximují parciální derivace, a všechny, až na Laplaceovy aproximují první derivace. Laplaceův aproximuje druhou derivaci.[10]

Pro příklad uveďme aplikaci Sobelova operátoru. Sobelova konvoluční maska pro x-ový a y-ový směr je dána maticí:

$$h_1 = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} h_2 = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}.$$

Výsledný výpočet gradientu v x-ovém a y-ovém směru s výpočtem velikosti hrany je zapsán jako:

$$gradX = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x-i, y-j) * h_1(i, j)$$

$$gradY = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x-i, y-j) * h_2(i, j)$$

$$g(i, j) = \sqrt{gradX^2 + gradY^2} ,$$

kde iterujeme pixely čtvercového okolí bodu a každou z hodnot pixelu násobíme hodnotou konvoluční masky na stejných souřadnicích. Tento postup je opakován v obou směrech, čímž je získána velikost hrany v daném směru. Z těchto hodnot je následně pomoci Pythagorovy věty vypočítána velikost celková. Hranový detektor nejdříve inicializuje konvoluční masky *h1* a *h2* pro horizontální a vertikální směr a následně je aplikuje na obraz tak, že osmi okolí bodu *x* vynásobí Sobelovým filtrem. Následně hodnoty pixelů po aplikaci filtru sečte. [3]

2.4 Prahování obrazu

Po aplikaci hranového filtru ještě není celý proces hledání hran dokončen. Bylo pouze vypočítáno, jakou hodnotu gradientu, tedy směru růstu, má každý pixel. Dále je potřeba toto pole hodnot gradientů projít a vyhledat vhodné kandidáty pro to, aby se staly hranovými pixely. Tomuto procesu hledání se říká prahování.

Prahování může být provedeno s hysterezí, což znamená, že budou určeny prahy dva. Práh T1 bude spodním prahem. Hodnoty gradientů dosahující pod hranici T1 jsou automaticky zařazeny jako hodnoty pozadí. Pixely s gradientem vyšším než je hodnota T2, jsou naopak okamžitě zařazeny jako hranové. Pro pixely s gradientem mezi těmito hranicemi je proveden algoritmus hledání sousedících hranových bodů, který eliminuje pixely s vyšší hodnotou gradientu, než je T1, které nesousedí s žádným jiným hranovým bodem.

Prahování s hysterezí má ale smysl pouze, pokud před samotným procesem prahování předchází takzvané ztenčování (nonmaximum suppression). Tento proces dělá přesně to, co říká doslovný překlad – potlačuje hodnoty, které nejsou lokálním maximem. Výsledkem je skutečné ztenčení hrany. Následným prahováním s hysterezí jsou odstraněny pixely, které jsou příliš slabé na to, aby se staly

hranovými. Převážně jde o šum. Druhou, jednodušší možností je využití jednoduchého prahování s jednou hodnotou T. Každý pixel s hodnotou gradientu vyšší než hranice T je určen jako hranový. Tento vztah je zachycen následující formulí: [3]

$$x_{i,j} = \begin{cases} x<T \rightarrow x=0 \\ x>T \rightarrow x=1 \end{cases}.$$

Jak již bylo popsáno, globální prahování je proces porovnávající všechny pixely s jednou hodnotou T. Získání adekvátní a dobře fungující hodnoty T je ale neméně důležitým úkolem, jako samotné spočítání velikostí gradientů. Hodnotu T lze získat jak vědecky, například využitím algoritmu Otsu [3], nebo nevědecky stylem pokus omyl. Tento postup může být vhodným v případě, že budeme chtít prahovat pokaždé obrazy s podobným histogramem.

Sofistikovanější metodou je využití zmíněného algoritmu Otsu. Ten je pojmenován po japonském vědci a matematikovi Nobyuki Otsuovi. Tato metoda je založena na myšlence, že po provedení prahování jsou všechny pixely zařazeny do 2 skupin. Buď do skupiny pixelů označujících objekt, v našem případě hranu, nebo do skupiny označující pozadí. Uvnitř skupiny by všechny pixely měly mít co možná nejvíce podobné hodnoty pixelů. Což z pohledu statistiky znamená malý vnitroskupinový rozptyl. Analogicky tedy platí, že mezi skupinový rozptyl bude co největší.

2.5 Morfologické operace

Z obrazu bylo předchozími kroky zjištěno, kde se nachází hranové pixely. Nicméně i přes veškerou snahu o potlačení veškerého šumu a nalezení skutečně všech čistě hranových pixelů se v obraze mohou vyskytovat nejasné malé objekty, které původně měly tvořit jeden celek, leč vlivem šumu a následným zpracováním byly rozděleny. Opakem může být případ, kdy v obraze bylo původně několik objektů, které byly vlivem poškození obrazu částečně slity do jednoho objektu.

V obou případech pomůže zpracování obrazu za pomoci morfologických operací. Matematická morfologie je technika používaná k analýze tvarů a vlastností obrazu. Tato operace prozkoumává obraz za pomoci takzvaného strukturního elementu, což je bodová množina s předdefinovaným tvarem. Ta je postupně umístěna na všechny pixely v obraze a je vždy porovnáván se sousedícími pixely.

Některé morfologické operace zkoumají, zda pixel odpovídá nebo neodpovídá lokálním tvarům v obraze. Jiné pravý opak.

Při problémech s nespojitými objekty v obraze, po provedení hranové detekce a binarizace obrazu, je nasnadě využití dilatace. Dilatace je dána vzorcem:

$$X \oplus B = \left\{ d \in E^2 : d = x + b, x \in X, b \in B \right\}$$
$$X \oplus B = \bigcup_{b \in B} X_b \quad ,$$

zde na množinu pixelů *X*, tedy obraz, aplikujeme strukturní element *B*. Prvky z obrazové množiny označujeme jako *x* a prvky ze strukturního elementu *b* a po aplikaci strukturního elementu dostáváme změněnou hodnotu *d*.

Nejdříve je obraz vertikálně i horizontálně rozšířen z každé strany kvůli šířce strukturního elementu. Následně je obraz procházen a pro okolí kontrolovaného pixelu je proveden logický součin se strukturním elementem, a je-li alespoň jedna hodnota rovna 1, pak je i pixelu přiřazena hodnota 1.

2.6 Segmentace

Na předzpracovaném obrazu je následně možné najít spojité oblasti pixelů s celkem velkou pravděpodobností, že pixely v jedné oblasti budou náležet právě jednomu objektu. Obraz je tedy možno celý projít a spojité oblasti si označit číslicí od 1 do n, kde n je celkový počet nalezených objektů v obraze.

Při druhém průchodu obrazem jsou získávány obdélníkové výseče definovány souřadnicí levého rohu, výškou a šířkou. Po zobrazení pouze této výseče je možné dostat obraz, který zobrazuje pouze jeden objekt a na kterém bude možno následně spustit algoritmus rozpoznávání.

3 Rozpoznávání strojovým učením

Schopnost čtenáře této bakalářské práce text číst, a následně mu i porozumět, je ve skutečnosti schopností jeho mozku a nervové soustavy jako celku. Myšlenka, že by bylo možné tomuto naučit i počítač, přiměla lid k vymýšlení postupů a algoritmů, jak toho dosáhnout.

Jako formální definici strojového učení lze využít definici Toma Mitchella: „Počítač se učí ze zkušenosti E, kterou získal při řešení problému T s přesností P, pokud se přesnost P s touto zkušeností zlepšuje.“ [11]. Pokud tuto definici aplikujeme na reálný příklad třídění příchozích emailů do spamu, pak T je právě klasifikace emailu jako spamu. Písmeno E zde substituuje zkušenost zařazení emailu jako spam a P říká, zda byl email zařazen správně, či nikoliv. Takže počítač se bude učit, pakliže prvně zařadil email jako spam, a ve skutečnosti se o spam nejednalo, a příště už tak neučiní.

Učící algoritmy lze rozřadit do dvou subkategorií – a to jako algoritmy učení s učitelem a algoritmy učení bez učitele. Učení s učitelem je pravděpodobně nejpoužívanější typ strojového učení. Algoritmu učení s učitelem jsou předány vzorky dat a k těmto vzorkům jejich správné výsledky. Algoritmus na základě těchto dat odvodí pravidla pro určení výsledné hodnoty dalšího příchozího vzorku. Tento postup lze použít například v medicíně. Pokud má algoritmus údaje o velikosti nádoru a o tom, zdali byl tumor zhoubný nebo nezhoubný, je tento algoritmus schopen s určitou mírou pravděpodobnosti na dalších příchozích snímcích tumorů určit, jedná-li se o tumor zhoubný a naopak.

V kontrastu s tímto postupem existují algoritmy učení bez učitele. Při tomto postupu jsou algoritmu poskytnuty pouze vzorky, na jejichž základě je pak algoritmus schopen klasifikovat vzorky na základě podobnosti. Jako příklad si lze představit obrovský agregační portál novinek, který bude procházet každý článek a na základě jeho obsahu říci, do které patří kategorie (sport, politika, informační technologie atd.). [4]

V této bakalářské práci bude využit ke strojovému učení algoritmus neuronových sítí inspirován fungováním mozku. Motivací může být výzkum na zvířecím mozku. Ten zjistil, že pokud přesměrujeme vjemy z oka do oblasti, kam za normálních okolností přichází vjemy z uší, tato oblast se dokáže naučit reagovat na tyto vjemy, tedy vidět. Což je velice pozoruhodné zjištění, z kterého lze odvodit možný postup pro řešení obtížných úloh vyžadujících přemýšlení. Místo implementace obtížných algoritmů je možné

naimplementovat algoritmus jeden. Algoritmus, který je schopen data uchovávat a učit se z nich. Algoritmus, kterému poskytneme trénovací množinu dat, již si lze představit jako dostatečně velkou množinu dat obsahující relevantní data pro naučení neuronové sítě.

To znamená, že v ní bude v dostatečném množství zastoupen každý z objektů, které chceme počítač naučit rozpoznávat. Ten pak bude schopen z takovéhoto vzorku odvodit pravidla. Pravidla budou následně použita na dalších vzorcích k odvozování, o který objekt se jedná. Čím více bude počítači poskytnuto vzorků k naučení těchto pravidel, tím větší přesnosti bude následně tento algoritmus dosahovat. [4]

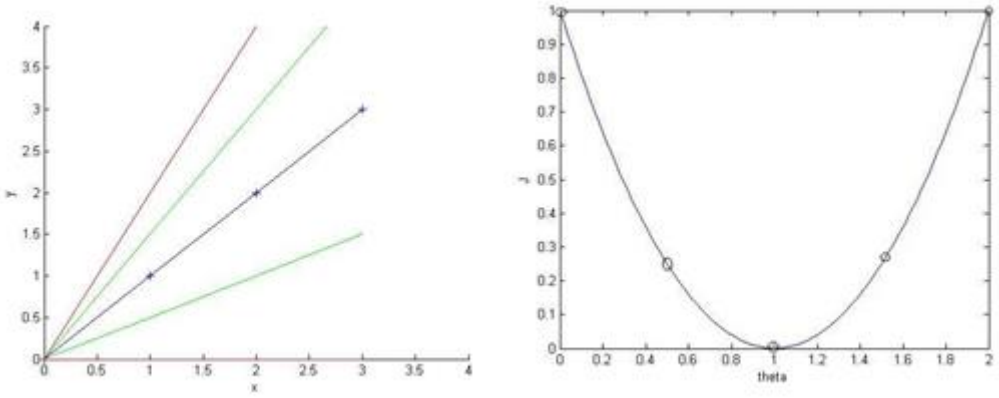
Neuronová síť se skládá z několika prvků. Těmi jsou model perceptron, který simuluje chování neuronu a váha jeho spojení s perceptronem na následující vrstvě – nazývána prostě váha spojení. Efektivita zvolených vah je udávána účelovou funkcí.

3.1 Účelová funkce

Nejprve je potřeba definovat účelovou funkci, což je konvexní funkce, která má za úkol shrnout efektivitu zvolených vah spojení na základě porovnání aktuálního výsledku s výsledkem očekávaným. Označme účelovou funkci J. V podstatě, pokud trénovací algoritmus odvedl dobrou práci a síť má zapamatované adekvátní hodnoty biasů a vah spojení. Hodnota účelové funkce $J \approx 0$ a analogicky, pokud jsou váhy a hodnoty biasů zvoleny nesprávně, hodnota funkce J bude vysoká. Cílem neuronové sítě při procesu učení je tedy postupnými kroky hodnotu této funkce minimalizovat, a to za pomoci algoritmu zvaného gradient descent. Správné zvolení předpisu účelové funkce je stěžejní pro celý proces následného učení.

Ve společnosti účelové funkce je třeba ještě definovat funkci hypotézy neboli aktivační funkce, která na základě vah spojení říká, jaká je aktuální výstupní hodnota. Nejjednodušší funkcí hypotézy je $h_\theta(x) = \theta x$, kde θ je vektor vah spojení obohacený o váhu biasu a x je vektor hodnot obohacený o 1. Takovýto předpis je nicméně již méně vhodný pro úlohy řešené pomocí neuronových sítí, té bude věnována následující část této kapitoly.

Jedním z nejjednodušších, leč účinných tvarů účelové funkce, může být takzvané počítání průměrné kvadratické chyby dle předpisu $J_\theta = \frac{1}{2m}\sum_x (h_\theta(x) - y)^2$, ve které sčítáme rozdíl mezi aktuální chybou hypotézy a správným výsledkem a tento subvýsledek následně dělíme hodnotou dvakrát počet prvků.



Obrázek: 3 Účelová funkce (levá část – funkce hypotézy pro θ, pravá část – účelová funkce)

Pro lepší představu jsou na Obrázek: 3 vidět vlevo funkce hypotézy, kde jsou vstupní prvky x (0, 1, 2, 3) závislé pouze na jednom parametru θ, který nabírá hodnot z leva 2, 1.5, 1, 0.5, 0. Jak je z obrázku možno vidět nejpřesnější hypotézou je funkce s hodnotou $\theta = 1$. Pro tyto hodnoty θ je následně vypočítána účelová funkce J, jejíž graf je vidět na obrázku vpravo. Pořadí zvýrazněných bodů odpovídá pořadí hodnot θ. Jak je z účelové funkce zřejmé, pokud je parametr nastaven správně, hodnota účelové funkce dosahuje hodnoty blízké nule. Tato ideální hodnota se dá získat pomocí algoritmu zvaného gradient descent.

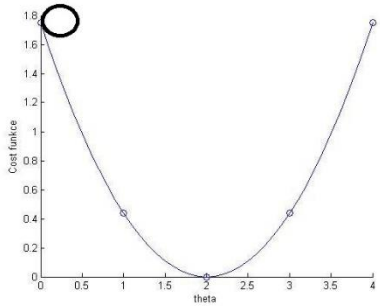
3.2 Gradient descent

Tento algoritmus slouží k postupnému zmenšování hodnoty účelové funkce. Jestliže máme danou funkci *J*, která je závislá na parametrech $\theta_0 \dots \theta_n$. To, co algoritmus dělá, je, že po malých částech mění hodnoty těchto parametrů tak, že velikost účelové funkce na nich závislé konverguje k nule. Poté, co bude dosaženo takovéhoto výsledku, je možné vzít sadu vstupních dat a na základě předpisu funkce hypotézy předpovědět, kam vstupní data zařadit.

Jedním z možných přístupů pro získání minima je zajisté výpočet derivací k získání bodů, ve kterých J dosahuje svého minima. To může být jednoduché v případech, jako je výše zakreslený graf Obrázek: 3, kde je funkce *J*, závislá pouze

na jednom parametru. Avšak v případech, kdy je účelová funkce závislá na vícero parametrech θ, je výpočet pomocí tohoto kalkulu značně obtížné. Nabízí se otázka, jaký jiný způsob tedy použít.

Pokud je počáteční hodnota parametru θ rovna 0, bude, jak je možno vidět na obrázku Obrázek: 3, hodnota účelové funkce vysoká. Představme si nyní, že do tohoto bodu umístíme kuličku. Ta dle zákonů fyziky zvolí směr k minimu funkce.



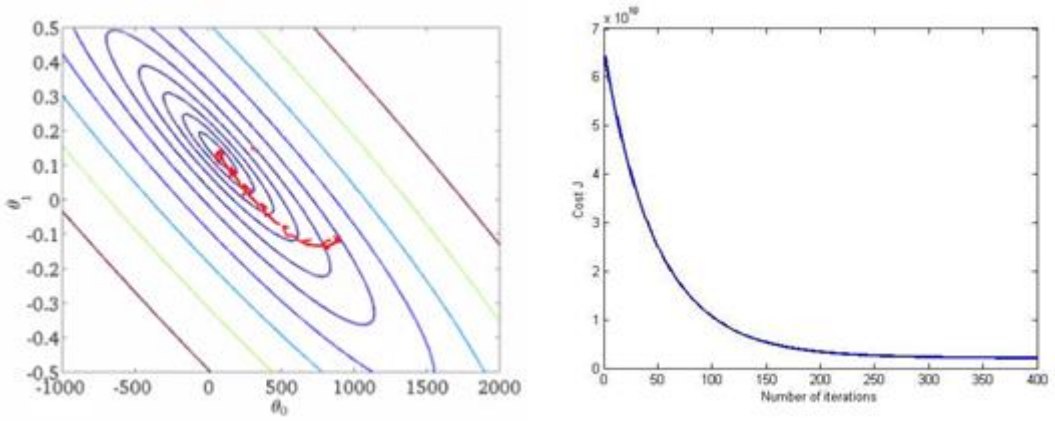
Obrázek: 4 Gradient descent

To, co kulička v každou chvíli opravdu dělá, je, že se posune o malý kousek $\Delta\theta$, ve směru θ. Kalkulus nám říká, že změna účelové funkce je rovna $\Delta J_\theta \approx \frac{\partial J}{\partial \theta} * \Delta\theta$. Jak je z tohoto výpočtu zřejmé: změna hodnoty účelové funkce je kromě změny hodnoty parametru θ závislá ještě na hodnotě $\frac{\partial J}{\partial \theta}$, kterou definujeme jako vektor gradientu a značíme ∇J. S touto znalostí lze vzorec přepsat na zápis $\Delta J_\theta \approx \nabla J * \Delta\theta$.

Z tohoto vzorce je následně možno odvodit, jak zvolit $\Delta\theta$ tak, aby bylo dosaženo negativního ΔJ_θ neboli zmenšení účelové funkce. Pokud totiž zvolíme $\Delta\theta = -\eta\nabla J$, kde η je zvaný parametr učení, pak zmíněný vzorec $\Delta J_\theta \approx \nabla J * \Delta\theta$ zaručuje, že po dosazení bude ΔJ_θ negativní. Tedy pokud se bude hodnota $\Delta\theta$ měnit vždy o $-\eta\nabla J$, bude J klesat. Z tohoto kalkulu je tedy možné odvodit pravidlo pro aktualizaci parametru $\theta : \theta \rightarrow \theta' = \theta - \eta\nabla J$.

Pro shrnutí gradient descent funguje tak, že spočítá gradient ∇J a pomyslnou kuličku posune v opačném směru. Průběh změny hodnoty účelové funkce při použití algoritmu Gradient descent je možné vidět na obrázku [4].

Pokud jednotlivé konkrétní hodnoty $\Delta\theta$ a ∇J nahradíme vektory $\Delta\theta = (\Delta\theta_1, \dots \Delta\theta_m)^T$ a $\nabla J = (\frac{\partial J}{\partial \theta_1}, \dots, \frac{\partial J}{\partial \theta_m})$, neboli místo jednoho parametru θ, bude účelová funkce závislá na vícero parametrech, bude vše fungovat naprosto stejně.



Obrázek: 5 Gradient descent – průběh[4]

3.3 Klasifikace

Problém rozpoznávání číslic lze obecně uchopit jako problém typu: je sada vstupních hodnot obrázku podobná se vstupem obrázku, na kterém se nacházela konkrétní číslice? Tuto podobnost matematicky procentuálně vyjadřuje právě výše zmíněná funkce hypotézy. Její definiční obor se však v klasifikačních problémech omezuje na hodnoty od 0 do 1 a jejím prahováním s číslicí 0,5 získáme diskrétní hodnoty znamenající, zdali se předložený vzorek shoduje nebo neshoduje s výsledkem. Klasifikační algoritmus je často také nazývaný algoritmem logistické regrese. [4]

Výše zmíněné prahování definuje matematická funkce pro výpočet rozhodovací hranice. Tu si lze na jednoduchém případě představit jako lineární funkci, pro kterou platí, že všechny hodnoty nacházející se nad touto hranící jsou klasifikovány jako spíše podobné množině vstupních hodnot. Tedy jsou klasifikovány jako objekty stejného typu. A označeny diskrétní hodnotou 1.[4]

3.4 Učící algoritmus klasifikace

Logistická regrese je v současnosti jedním z nejpoužívanějších učících algoritmů. [4] O to se zapříčiňuje mimo jiné i hojné využití tohoto učícího modelu v neuronových sítích, což je algoritmus, kterému se tato práce bude podrobněji věnovat dále. Zatímco výstupem předpovědi u prvně zmíněné účelové funkce bylo libovolné číslo na spojitém oboru hodnot, u logistické regrese je výsledkem právě hodnota 1 nebo 0. To je totiž opisný tvar pro skutečný význam těchto dvou číslic, a to: 1 pro náleží do klasifikované třídy nebo 0 pro nenáleží. Například má pacient tumor zhoubný, nebo nezhoubný? Je příchozí email spam, nebo není?

Výstupem logistické regrese je hodnota 0 nebo 1, nicméně skutečným výstupem fyzického neuronu v neuronové síti je hodnota na spojité škále od 0 do 1. Tato hodnota je následně prahována a prahováním získáváme diskrétní hodnoty. Je tedy zapotřebí zavést formu funkce hypotézy, která bude mít definiční obor ve zmíněném rozpětí, neboť funkce používaná u lineární regrese dosahuje hodnot i daleko větších, než je požadovaná horní hranice, i daleko menších než spodní hranice. Z předchozího předpisu je však možno vycházet a její hodnoty pouze normalizovat na hodnoty v definičním oboru. Toho je dosaženo pomocí takzvané sigmoid funkce neboli logistické funkce. Kterou definujeme jako $g(z)$, ve které z substituuje původní předpis funkce a funkce g má tvar:

$$g(z) = \frac{1}{1+e^{-z}}.$$

Tu je možné interpretovat jako pravděpodobnost, že výstupní hodnota bude rovna 1, tedy:

$$h_\theta(x) = g(z) = p(y = 1|x; \theta),$$

neboli pravděpodobnost, že při vstupní hodnotě x nabude hodnoty 1, přičemž pravděpodobnost je parametrizovaná parametrem nebo parametry θ.[8]

Ještě zbývá definovat účelovou funkci. Problémem je však nelinearita použité sigmoid funkce. Prvně definovanou účelovou funkci

$$J_\theta = \frac{1}{2m}\sum_x (h_\theta(x) - y)^2 ,$$

můžeme přepsat na tvar:

$$\frac{1}{m}\sum_{i=1}^{m} cost(h_\theta(x), y),$$

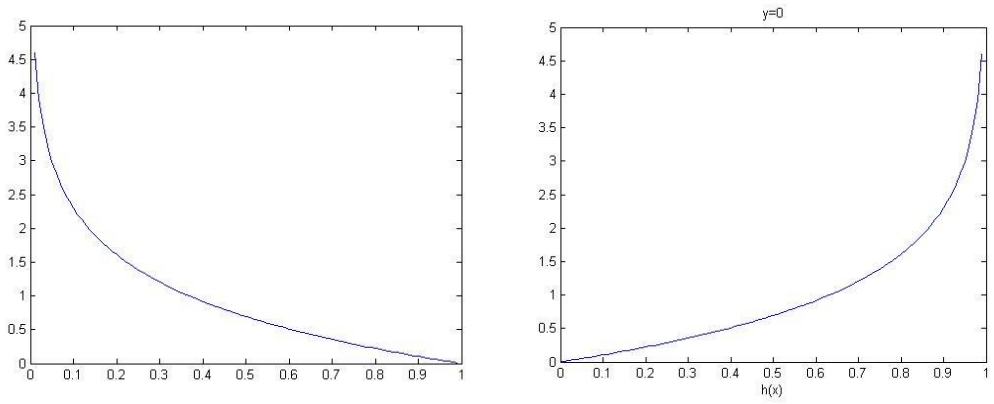
kde v tomto případě je

$$cost(h_\theta(x), y) = \frac{1}{2}(h_\theta(x) - y)^2.$$

Při dosazení sigmoid funkce do účelové dostaneme nekonvexní funkci. To ztěžuje celý problém hledání globálního minima jakožto ideální hodnoty natrénované sítě. Proto je potřeba definovat funkci jinou, která bude splňovat předpoklad konvexnosti. $cost(h_\theta(x), y)$ je tedy nutné nahradit jinou, která ve spojení se sigmoid funkcí bude konvexní. [8] Tuto podmínku splňuje logaritmická funkce:

$$cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) \ pro\ y = 1 \\ -\log(1 - h_\theta(x)) \ pro\ y = 0 \end{cases}.$$

Pro levý graf na Obrázek: 6, tedy pokud y = 1 platí, že pokud se hodnota funkce hypotézy přibližuje 1, cost funkce je rovna 0, a analogicky naopak. Pokud je funkce hypotézy rovna 0, pak hodnota cost funkce se blíží nekonečnu. Tedy pokud výsledek hypotézy neboli predikce říká, že výstupem má být hodnota 0, ale ve skutečnosti je správným výstupem hodnota 1, je nutné učící algoritmus penalizovat (upravit). Analogicky stejná pravidla aplikujeme na případ, kdy y = 0. [4]



Obrázek: 6 Logaritmická funkce pro y=0 a y=1,

Tuto formu podmíněného dvouřádkového zápisu účelové funkce lze zapsat jako:

$$Cost(h_\theta, y) = -y * \log(h_\theta(x)) - (1 - y) * \log(1 - h_\theta(x)).$$

Toto lze učinit díky tomu, že pokud y bude nabývat hodnoty 0, pak první člen rovnice, tedy:

$$-y * \log(h_\theta(x)),$$

bude roven 0, a pokud y = 1, pak:

$$(1-y)*\log(1-h_\theta(x))$$

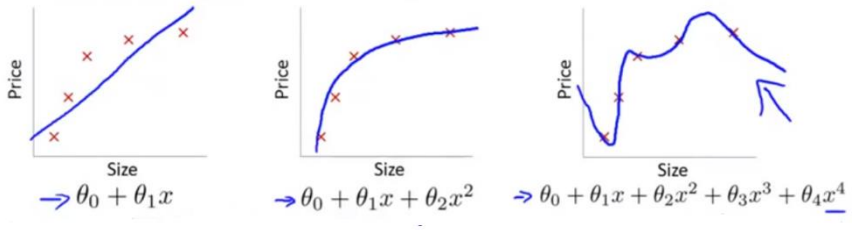
je roven 0. Tato cost funkce vychází z metody maximální věrohodnosti, která označuje jednu z centrálních metod matematické statistiky. Z této cost funkce vychází účelová funkce používaná v logistické regresy. Definujeme ji tedy jako:

$$J_\theta = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^i) + (1-y^{(i)}) \log(1-h_\theta(x^i))\right].$$

Z této účelové funkce lze stejně jako z prvně definované určit pravidlo, na jehož základě budou měněny hodnoty parametrů θ tak, že celková hodnota účelové funkce bude klesat, až dojde k minimalizování hodnoty účelové funkce k hodnotě blízké 0. Toto pravidlo jde stejně jako v případě lineární regrese definovat jako opakování kroků: $\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$, ve kterém je kalkulus pro $\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^i) - y^i\right) * x_j^i$.

3.5 Regularizace parametrů

Téma regularizace parametrů, a tedy celé účelové funkce, je úzce spjaté s problémem přeučení. Přeučení se v učícím algoritmu může objevit, pakliže naše funkce hypotézy h_θ je příliš přizpůsobena trénovací množině dat. Učící algoritmus následně není schopný generalizace problému a selhává v rozpoznávání dalších příkladů, které nejsou součástí trénovací množiny. Na obrázku je problém přeučení znázorněn vpravo, vlevo je znázorněn jev opačný, kdy funkce hypotézy nedostatečně přesně odpovídá skutečnosti, a v důsledku toho pak dochází k velké chybovosti. Ideální funkce hypotézy je znázorněna uprostřed.



Obrázek: 7 Regularizace parametrů [4]

Úkolem regularizace je udržet parametry, na kterých je funkce hypotézy závislá, co nejmenší pomocí penalizace. K účelové funkci je tedy nutné přidat část rovnice, která se postará o udržení parametrů dostatečně malých:

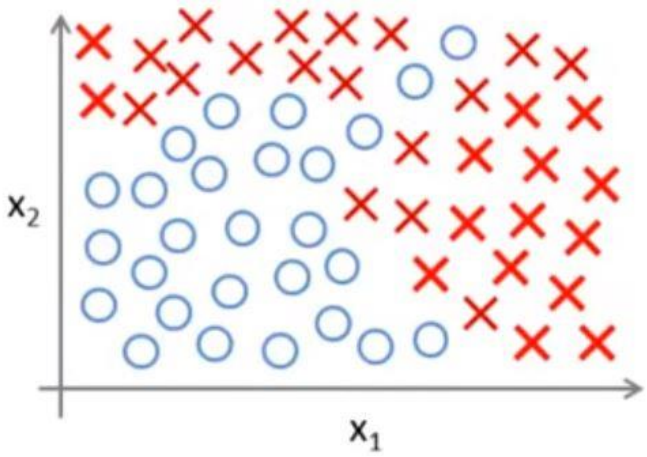
$$J_\theta = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^i) + \left(1 - y^{(i)}\right) \log(1 - h_\theta(x^i))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2,$$

kde L je počet vrstev, S_l počet neuronů vrstvy a λ je regularizační parametr, který vyvažuje poměr mezi udržením malých hodnot parametrů a dostatečného přizpůsobení účelové funkce trénovací množině.

Tímto dojde k vyhlazení grafu funkce, a algoritmus předpovídání je pak schopen problém zobecnit a dosahovat lepších výsledků.

3.6 Problém složité rozhodovací hranice

Zřídka kdy bude rozhodovací hranice mít jednoduchý lineární charakter. Pakliže nemá, je zapotřebí do funkce hypotézy zavést složitější nelineární sledované znaky. Řekněme, že chceme predikovat výsledek, který je závislý na 2 neznámých x_1 a x_2, a k dostání je historie výsledků:



Obrázek: 8 Historie výsledků, u kterých není lineární rozhodovací hranice.

Pokud aplikujeme logistickou regresi s velkým množstvím polynomů skládajících se z kombinací těchto proměnných $g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_3 x_1 x_2^2 + \cdots)$, získáme rozhodovací hranici, na jejímž základě budeme moci přesně odhadnout výsledek. Tato metoda může fungovat, pokud máme skutečně jen 2 proměnné, na kterých výsledek závisí. Pokud je těchto proměnných

100, pak (pokud se omezíme na kvadratický stupeň) získáváme zhruba 5 000 polynomů, a v případě kubického stupně je polynomů okolo 170 000.

V případě úlohy rozpoznávání číslic počítač nedokáže vidět číslice jako člověk, je potřeba mu informaci o obrázku nějakým způsobem předat. Definujme v tomto vztahu pojem rys, který symbolizuje informaci. Počítači jsou tedy předány tyto rysy, na jejichž základě se musí rozhodnout. Nejpřirozenějším rysem obrazové informace je intenzita pixelů v něm obsažených. Pokud tedy počítači předáváme rysy obrazu o velikosti 50 x 50 ve formě vektoru, předáme mu celkem 2 500 rysů obrazu, pokud se omezíme na obraz v šedo tónu.

Pokud chceme předat obraz v RGB, předáváme informaci o každém kanále, takže 2 500*3. Pokud následně z těchto rysů uspořádáme funkce hypotézy s kvadratickými polynomy, kombinací těchto hodnot získáváme 3 miliony těchto polinomů. Což je pro počítač velmi složité spočítat. Je tedy zapotřebí vyřešit, jak uchopit problém se složitou rozhodovací hranicí. K tomuto poslouží algoritmus neuronových sítí.

3.7 Neuronová síť

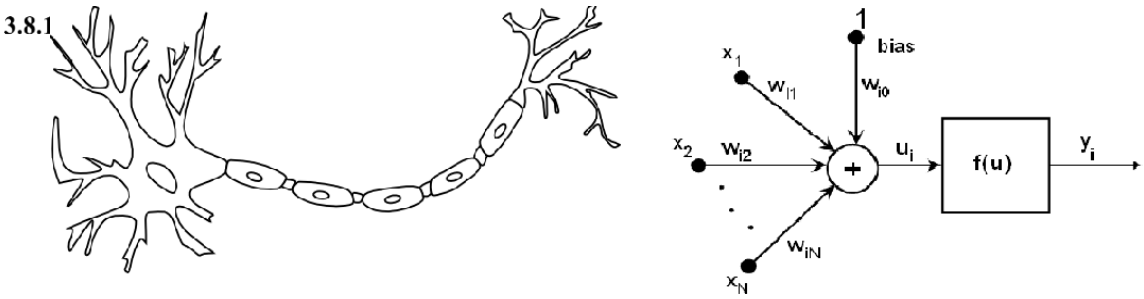
Neuronová síť (dále jen NS) je abstraktní počítačový model lidského mozku. Mozek má přibližně 10^{11} neuronů, což jsou základní stavební kameny mozku. Neurony jsou spolu provázány zhruba 10^{15} vazbami (synapsemi). Na základě lidského mozku byl navrhnut model NS, který se skládá také z neuronů a spojení. Pokud je takováto NS pak převedena do grafu, neurony jsou reprezentovány jako vrcholy a spojení jsou jako hrany. [4]

3.8 Neuron

Stejně tak jako je neuron základním stavebním kamenem lidského mozku, je také hlavním konstruktem NS v počítačové oblasti. Na Obrázek: 9 je vlevo lidský neuron a vpravo simulovaný neuron v neuronové síti.

3.8.1 Neuron v mozku

Tato buňka je schopna nést, zpracovávat a přimět organizmus reagovat na



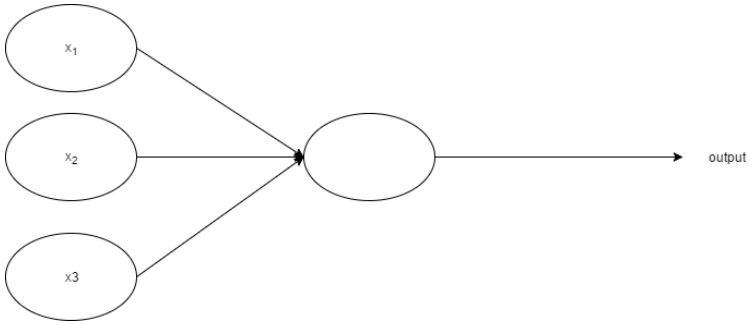
Obrázek: 9 Neuron a simulovaný neuron [12]

podněty. Jeden neuron se poté dělí na dendrity, které jsou dostředivé a slouží pro příjem vzruchů do jádra neuronu. Dále jádro neuronu, kde je rozhodnuto, zda se na vzruchy bude, či nebude reagovat, a v neposlední řadě odstředivý axon. Ten slouží pro spojení s ostatními částmi nervové soustavy. [5]

3.8.2 Simulovaný neuron

První umělé neurony zvané perceptrony vznikaly v 50–60. letech pod křídly vědce Franka Rosenblatta, který byl inspirován pány McCullochem a Pittsem. Perceptron na Obrázek: 10 přijímá binární data. Na tomto příkladu jsou neuronu poskytnuta data x_1-x_3, z nichž produkuje jedinou výslednou hodnotu.

Výsledná hodnota je vypočtena na základě vah spojení, které jsou označovány



Obrázek: 10 Perceptron

malým písmenem *w* z anglického slova weight a označují důležitost daného spojení.

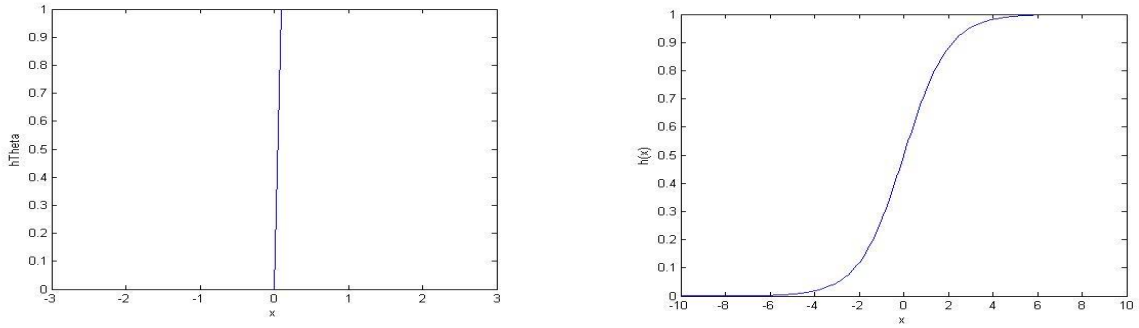
Jak již bylo zmíněno Výsledkem je jediná hodnota, která je dána váženou sumarizací hodnot x, tedy $\sum_j w_j x_j$. Tato sumarizace je následně porovnána s prahovou hodnotou. Pakliže je menší než hodnota prahu, je výsledek roven 1, a analogicky naopak $\begin{cases} 0\ pokud\ \sum_j w_j x_j \leq práh \\ 1\ pokud\ \sum_j w_j x_j > práh \end{cases}$. Tento zápis je nicméně zdlouhavý a nepřehledný, pro zjednodušení je možné využít vektory. Vektor w symbolizující váhy spojení a x symbolizující vstupní hodnoty, a výsledná hodnota je pak skalárním součinem těchto dvou vektorů $\sum_j w_j x_j = w.x$. Další změnou pro zjednodušení výpočtu je přesunutí prahu z pravé strany rovnice na stranu levou.

Tímto přesunem definujeme hodnotu zvanou **bias**, která má hodnotu $b = -práh$. Výpočetní vzorec výsledné výstupní hodnoty je tedy konečně zformulován jako $\begin{cases} 0\ pokud\ w.x + b \leq 0 \\ 1\ pokud\ w.x + b > 0 \end{cases}$. Bias tedy v podstatě vyjadřuje hodnotu toho, jak snadné je, aby konečný výsledek byl roven 1 neboli biologicky přesněji, aby perceptron zareagoval.

V současnosti je model neuronu perceptron nahrazený modelem sigmoid. Ten se navenek jeví obdobně jako model perceptronu. Nicméně, jeho vnitřní chování je lehce odlišné. Stejně jako u perceptronu je vstupní hodnotou vektor x, výslednou hodnotou však není právě 1 nebo 0, ale jakákoliv hodnota v rozmezí těchto hodnot včetně jich samých. Z původního předpisu perceptronu je odvozen vzorec sigmoid neuronu $\sigma(w.x + b)$, kde σ je nazývána jako sigmoid funkce. Pro tuto funkci je typický její S-ovitý tvar. Definičním oborem jsou buď hodnoty od -1 do 1, nebo častěji (hlavně v kontextu neuronových sítí) od 0 do 1. Krajním hodnotám se tato funkce blíží limitně. Nejčastěji je používán tento předpis $\sigma(z) = \frac{1}{1+e^{-z}}$. [8].

Podobnost modelu sigmoidu a perceptronu je značná. Pro případy, kdy je výsledkem funkce $w.x + b$ velmi vysoké číslo, by výstupní hodnota perceptronu nabývala hodnoty 1, stejně tak je tomu u sigmoidu, neboť při velkém $w.x + b$ platí, že $e^{-z} \approx 0\ a\ \sigma(z) \approx 1$. Tedy pro dostatečně velké pozitivní $w.x + b$ je výstupní hodnota aproximována k 1 a analogicky pro velké negativní číslo. Rozdíl je tedy pouze v hodnotách průměrných. Jak je vidět na následujících funkcích modelů. Obrázek: 11

Nutnost vyhlazené sigmoid funkce bude vysvětlena v následujících kapitolách.

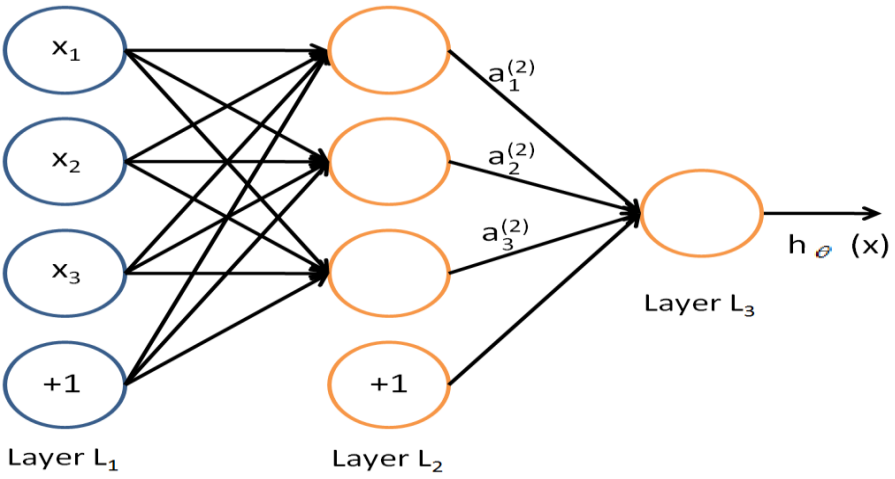


Obrázek: 11 Funkce perceptronu a sigmoidu

3.8.3 Model sítě

Neuronová síť se v lidském těle skládá z jednotlivých neuronů, které spolu kooperují k dosažení jednotného cíle. Stejně tak je tomu i ve výpočetním modelu počítačové neuronové sítě. Tak jako jedna mozková buňka nemůže porozumět ničemu, nemůže být ani perceptron, potažmo sigmoid, schopný sám zpracovávat obtížnější úkony. Pro složitější úkony je nutné dohromady spojit vícero neuronů.

Na Obrázek: 12 se nachází jedna vstupní vrstva, jedna skrytá vrstva a jedna vrstva výstupní. Mezi nimi se nachází jedna vrstva, která nese označení skrytá. Každý simulovaný neuron na vstupní vrstvě nedělá nic jiného, než že zváží výstupní hodnoty a ty následně přeposílá další, v tomto případě skryté vrstvě. Na skryté vrstvě již dochází k rozhodování na vyšší formě abstrakce, neboť vstupní hodnotou již není konkrétní hodnota pixelu, ale výsledek rozhodování neuronu na vrstvě první.



Obrázek: 12 Neuronová síť – výpočetní model

Komplexnost a vyšší míra abstrakce pak přibývá s každou další vrstvou. Výstupem každého dílčího neuronu je jedna hodnota, což se může zdát v rozporu s přiloženým obrázkem. Nicméně výstupem je opravdu jedna hodnota, která je dále rozeslána na všechny neurony sousední vrstvy.

Úkolem neuronové sítě je vypočítat hodnotu funkce H_θ a na základě této hodnoty rozhodnout, zda objekt, který je definován vstupními hodnotami x_1 až x_3, zařadit jako vyhovující nebo nevyhovující. Tohoto je dosaženo pomocí takzvaného procesu učení. Učení znamená schopnost neuronové sítě zapamatovat si konkrétní váhy spojení a hodnoty biasu a při chybovém rozhodnutí sítě tyto hodnoty mírně upravovat tak, aby se výsledek také o málo změnil a iterací po malých změnách dosáhl správných hodnot a správného rozpoznání objektu. Pokud tedy prvotní hodnotu váhy upravíme o malou změnu Δw, pak očekávaný výsledek je prvotní výsledek plus malá hodnota Δ*výsledek*. Pokud se tomuto skutečně děje neuronová síť po malých krůčcích mění hodnoty vah, až nakonec správný výsledek najde.

Problém ale nastává v případě, že máme v neuronové síti perceptrony. Ty mají schopnost nabývat hodnot pouze 0 nebo 1, tudíž i mála změna váhy může zapříčinit skokový rozdíl výsledné hodnoty. A tato malá změna pak kompletně změní chování celé sítě. To je důvod, proč byly perceptrony nahrazeny sigmoidy, které jsou prakticky stejné, nicméně malá změna vah spojení zapříčiní skutečně jen malou změnu výsledku. Neboť jak je možné vidět na grafech funkcí Obrázek: 11 aplikací sigmoid funkce σ získáme vyhlazenou verzi skokové funkce. A právě tato vyhlazenost je klíčová pro to, že po malé změně vah se výsledná hodnota změní také pouze o málo. Výpočet nám prozrazuje, že změna výsledku může být aproximována následovně: $\Delta výsledek = \sum_j \frac{\partial výsledek}{\partial w_j} \Delta w_j + \frac{\partial výsledek}{\partial b} \Delta b$, kde w je zástupný symbol pro vektor vah a b pro bias. [5,6,8]

To, jak celkově neuronová síť vypadá na vstupní a výstupní vrstvě, je ve většině případu zřejmé. U rozpoznávání konkrétních objektů bude mít neuronová síť na vstupu počet neuronů shodných s celkovým počtem pixelů v obraze. V obraze o rozměrech 20 x 20 bude tedy 400 vstupních neuronů. Na výstupní vrstvě je pak neuron jeden. Jeho hodnota rozhodne o tom, zda objekt bude nebo nebude posouzen jako typ objektu, s kterým byl právě porovnáván. Co je již méně zřejmé, je počet neuronů na skryté vrstvě a také počet skrytých vrstev samotných. Pravidla toho, jak by měla

vypadat skrytá vrstva a kolik skrytých vrstev by neuronová síť měla obsahovat, prakticky neexistují. Byla popsána pouze heuristická řešení pro požadovaná chování neuronové sítě.[4] Z jednoho heuristického řešení bude vycházet také neuronová síť sestrojena v této práci.

Ve spojitosti s vícevrstvou architekturou je nezbytné synchronizovat použité názvosloví a indexaci. Pro aktivační funkci i-tého neuronu na j-té vrstvě bude využit zástupný symbol $a_i^{(j)}$. Pro vektor vah spojení z j-té vrstvy na vrstvu j+1 symbol $\theta^{(j)}$. Pokud tedy budeme uvažovat například neuronovou síť, která bude mít na vstupu i ve skryté vrstvě právě 3 neurony a jeden výstupní, kalkulus pro výpočet aktivačních funkcí na základě hodnot ze vstupní vrstvy na první neuron skryté vrstvy říká:

$$a_1^{(2)} = g\left(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3\right).$$

Tedy proneuron skryté vrstvy získáváme hodnotu jeho aktivační funkce součtem násobků vah spojení vedoucích ze vstupní vrstvy k neuronu na vrstvě skryté ($\theta_{10}^{(1)}$ ... $\theta_{13}^{(1)}$). Stejný je následný postup pro zbylé neurony a výsledky aktivačních funkcí budou poskytnuty jako vstupy pro vrstvu výstupní. Pro tuto vrstvuje pak aplikován výpočet:

$$h_\theta(x) = a_1^{(3)} = g\left(\theta_{10}^{(2)} a_0 + \theta_{11}^{(2)} a_1 + \theta_{12}^{(2)} a_2 + \theta_{13}^{(2)} a_3\right).$$

Ten jako svou vstupní vrstvu používá výše získané hodnoty aktivačních funkcí neuronů na skryté vrstvě. Tímto výpočtem jsme získali predikci neuronové sítě. Aplikovanému postupu se podle toho, co dělá, říká dopředná propagace. Vstupní hodnoty jsou vzaty, na skryté vrstvě je jejich multiplikací s jejich váhami získána hodnota aktivační funkce každého neuronu, a tyto výsledné hodnoty vrstvy jsou následně propagovány na vrstvu další.

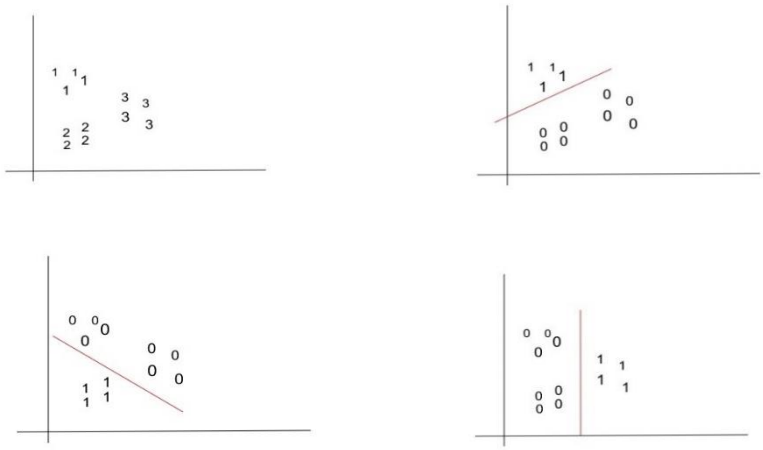
Při použití vektorizovaného řešení tohoto problému bude parametr aktivační funkce nahrazen symbolem z, tedy $a_1^{(2)} = g\left(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3\right)$ je ekvivalentní se zápisem $a_1^{(2)} = g\left(z_1^{(2)}\right)$ a analogicky pro ostatní neurony. Tyto parametry aktivačních funkcí lze následně zapsat jako vektor. Pro druhou vrstvu tedy platí $z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$. Vektorizovaně lze tedy výpočet aktivačních funkcí na skryté

vrstvě zapsat jako $z^{(2)} = \theta^{(1)}x$ a $a^{(2)} = g\left(z^{(2)}\right)$. Na výstupní vrstvě pak dostáváme vztah $z^{(3)} = \theta^{(2)}x$ a $h_\theta(x) = a^{(3)} = g\left(z^{(3)}\right)$. Pokud se nyní omezíme pouze na závislost výstupní vrstvy na vrstvě skryté, jde o klasickou lineární regresi definovanou a představenou v kapitolách 3.1 a 3.2. Co je ale nové, a co je právě důvodem toho, že algoritmy neuronových sítí jsou schopny řešit problémy složité nelineární rozhodovací hranice, je to, že hodnoty, které pro výstupní vrstvu slouží jako vstupní, jsou dány aktivační funkcí neuronu na skryté vrstvě, a tato aktivační funkce je parametrizovaná naučenou hodnotou váhy spojení se vstupní vrstvou.

V této práci bude využita architektura sítě založena na neuronové síti představené vědcem Anrew Ng, kterou představil na portále Coursera.org [4]. Skládá se ze 400 vstupních neuronů, což odpovídá obrázkům s číslicemi o rozměrech 20 x 20 a bude reprezentována 400prvkovým vektorem *x*. Na vrstvě výstupní je neuronů 10. 1 neuron pro každou z číslic 10 soustavy. Každý neuron na výstupní vrstvě bude dosahovat právě hodnoty 1, pokud je objekt klasifikován jako objekt, který je v síti zastoupen neuronem, nebo hodnoty 0, pokud tak objekt klasifikován není. Reprezentován bude 10prvkovým vektorem *y*. Na skryté vrstvě bude umístěno 25 neuronů.

3.8.4 Více třídní klasifikace

Architekturu a následný výpočet zmíněný výše v této práci je možné použít jen za předpokladu, že konečný výsledek bude pouze: ano, patří do třídy nebo ne, do třídy nepatří.



Obrázek: 13 Algoritmus jeden proti všem

To ale neodpovídá zadání neuronové sítě rozpoznávající čísla od 0 do 9, neboť počet možných výsledků je právě 10, a to *y=0*, *y=1*, …, *y=9*. Pro převedení tohoto problému na problém binární, tedy náleží nebo nenáleží do třídy, je možné využít algoritmu jeden proti všem. Ten je v grafu znázorněn na Obrázek: 13, který zachycuje rozdělení grafu se 3 skupinami do 3 různých grafů.

Na každém z těchto grafů je aplikován jeden proti všem přístup, tedy na pravém horním obrázku je za výslednou hodnotu označena skupina 1 a ostatní skupiny jsou označeny jako chybné. Z tohoto předpokladu je následně možné vycházet a určit rozhodovací hranici (znázorněna červenou barvou). Analogicky je postupováno s ostatními skupinami. Výsledek je v matematickém vyjádření možno zachytit výsledkovým vektorem. Pro případ 10 číslic definujeme 10 výsledkových vektorů následujícím způsobem: pro výsledek, který bude odpovídat hodnotě 5, definuje vektor (0, 0, 0, 0, 0, 1, 0, 0, 0, 0). Celkovou výsledkovou matici definujeme:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix}, \text{ tedy } Y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Na výstupní vrstvě neuronové sítě bude tedy 10 výstupních neuronů, z nichž pro každá vstupní data bude právě jeden dosahovat hodnotu 1 a ostatní 0.

3.9 Účelová funkce neuronové sítě

Stejně jako tomu bylo u již zmíněných učících algoritmů i algoritmus neuronových sítí k učení potřebuje formulaci vyjádření stavu svého momentálního naučení, a i zde je tato formulace zastoupena účelovou funkcí. Matematický model vychází z předpisu funkce logistické regrese. Tedy z matematické formule $J_\theta = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta\left(x^i\right) + (1 - y^{(i)}) \log(1 - h_\theta\left(x^i\right))\right]$. Je nutné si uvědomit, že neuronová síť pro rozpoznávání čísel bude řešit problém typu jeden proti všem. Tedy, že počet možných výsledků neuronové sítě bude pro jeden příklad právě 10. Účelovou funkci budeme počítat pro všech 10 výsledků a jejich výsledky sečte. Matematická formulace tedy zní:

$$J_\theta = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y^{(i)} \log h_\theta\left(x^i\right)_k + \left(1 - y_k^{(i)}\right) \log(1 - h_\theta\left(x^i\right))_k\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\theta_{ji}^{(l)}\right),$$

kde K je právě oněch 10 možných výsledků. Druhým sčítaným polynomem je regularizační parametr.

Algoritmus učení je dle jeho definice na základě zkušenosti, kterou je v případě rozpoznávání číslic poskytnutá sada trénovacích čísel, schopen změřit svoji výkonnost na základě spočtení účelové funkce a z tohoto měřítka pak vyvozovat změnu svých parametrů tak, aby docházelo ke zlepšování výsledků, tedy minimalizaci účelové funkce. Onu minimalizaci v předchozích případech strojového učení zastupoval algoritmus gradient descent. Nebude tomu jinak ani u neuronových sítí. Tedy hlavní myšlenka zůstává stejná, pouze přibývají vrstvy neuronové sítě. Pro shrnutí: potřebujeme měnit hodnoty vazeb θ na základě výpočtu $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$.

Postup pro získání těchto hodnot je nicméně lehce odlišný. Nejprve je třeba dopřednou propagací vypočítat účelovou funkci a na základě této hodnoty distribuovat chybu δ do všech neuronů *j* na vrstvách *l*. Tato dílčí chyba je označována jako $\delta_j^{(l)}$. Pro výstupní chybu na výstupní vrstvě definujeme vztah $\delta_j^{(L)} = a_j^{(L)} - y_j$, kde L je celkový počet vrstev v neuronové síti a $a_j^{(L)}$ je $\left(h_\theta(x)\right)_j$, tedy funkce hypotézy na výstupů vrstvě.

Pokud tento výpočet uvažujeme ve vektorové matematice, pak lze výpočet přepsat jako $\delta^{(L)} = a^{(L)} - y$. Hodnotu $\delta^{(L)}$následně propagujeme zpět do hlubších vrstev, tedy směrem ke vstupní vrstvě, za pomoci vzorce $\delta^{(l)} = \left(\theta^{(l)}\right)^T \delta^{(l+1)} .* g'\left(z^{(l)}\right)$, pokud l uvažujeme jako proměnnou s hodnotou rovnou pořadí řešené vrstvy. Výraz $g'\left(z^{(l)}\right)$ zde zastupuje derivaci sigmoid funkce vstupních hodnot l-té vrstvy. Tuto derivaci lze vyjádřit jako $a^{(l)} .* \left(1 - a^{(l)}\right)$, tedy Hadmardův součin vektoru aktivačních funkcí s vektorem *1* – vektor aktivačních funkcí vrstvy *l*. Chybu δ nepočítáme pro první vstupní vrstvu, neboť pro ni nemá smysl.

S vypočtenou hodnotou δ můžeme přistoupit k výpočtu parciální darivace účelové funkce podle $\theta_{ij}^{(l)}$ na základě vzorce $\frac{\partial}{\partial \theta_{ij}^{(l)}} = a_j^{(l)} \delta_l^{l+1}$. Ve spojitosti s výpočtem s agregováním hodnoty parciální derivace při učení neuronové sítě na *m* vzorcích zavádíme proměnnou Δ. Což je matice o rozměrech stejných jako matice vah spojení. Do ní před spuštěním algoritmu uložíme hodnotu 0 a s každým průchodem

aktualizujeme hodnotu $\Delta_{ij}^{(l)}$ o hodnotu $\frac{\partial}{\partial \theta_{ij}^{(l)}}$ neboli $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{l+1}$. Pokud budeme $\Delta_{ij}^{(l)}$ uvažovat jako prvek matice agregovaných chyb na vrstvě l, pak celá tato matice má zástupný znak $\Delta^{(l)}$ a výpočet lze přepsat jako $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}\left(a^{(l)}\right)^T$. Po zpracování posledního vzorku můžeme překročit k výpočtu hodnoty $D_{ij}^{(l)}$. Ta je rovna $\begin{cases} D_{ij}^{(l)} := D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} \; pro \; j \neq 0 \\ D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\theta_{ij}^{(l)} \; pro \; j = 0 \end{cases}$, tedy spodnímu výrazu pro všechny neurony vrstvy, kromě neuronu biasu. Tato hodnota je následně použita v algoritmu gradient descent případně v jiném, obdobném optimalizačním algoritmu. [4] [6] [7]

4 Implementace

4.1 Matlab

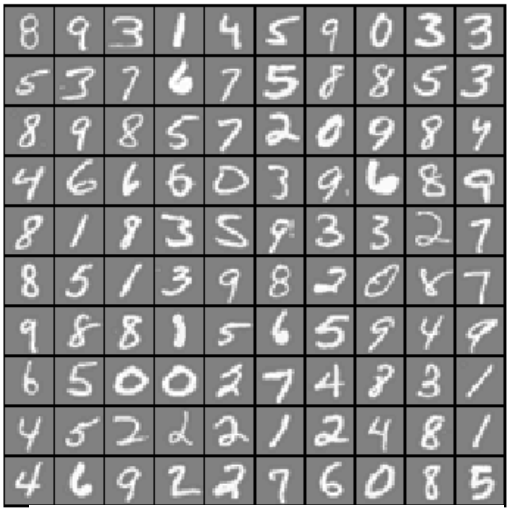
Pro implementaci řešení problému rozpoznávání číslic byl vybrán programovací jazyk Matlab. Hlavními důvody pro tento programovací jazyk jsou jednoduché práce s maticemi a maticovými počty. Dále pak snadné a rychlé prototypování, a v neposlední řadě intuitivnost jazyka jako takového a celého ekosystému (IDE, příkazová řádka, ladicí možnosti, tutoriály, dokumentace.

4.2 Testování řešení

Při implementaci algoritmu neuronových sítí bylo potřeba vyřešit několik problémů. První v řadě byl kde sehnat trénovací/testovací data. To nakonec bylo vyřešeno díky portálu Coursera.org [4]. Pro natrénování neuronové sítě a následné testování nabízí 5 000 příkladových obrázků ve formě matice o rozměru 20 x 20, normalizovaných na tvar 1*400 s hodnotami na škále od -1 do 1. K těmto hodnotám je poskytnuta i výsledná číslice, kterou by správně naučená neuronová síť měla predikovat. Vizualizovaná data je možné vidět na: 0Obrázek: 14. Tato matice dat je nahrána pomocí klíčového slova load, kterému je jako parametr předána cesta k souboru:

```
load('data.mat');
% Nahodny vyber 100 obrazku
sel = randperm(size(X, 1));
sel = sel(1:100);
```

```
zobrazeniDat(X(sel, :));
```



Obrázek: 14 Vizualizace trénovacích dat

Dále je potřeba definovat tvar neuronové sítě:

```
input_layer_size  = 400;  % 20x20 vstupnich pixelu obrazku
 hidden_layer_size = 25;   % 25 neuronu na skryte vrstve
num_labels = 10;          % 10 moznosti vystupu (10 cisel).
```

A následně jsou data předložena učícímu algoritmu:

```
[nn_params, cost] = fmincg(@(p)costFunkce(p,input_layer_size,
hidden_layer_size,num_labels, X, y, lambda),initial_nn_params, options);
```

ten na základě principů popsaných v této práci aktualizuje hodnoty vah spojení, které jsou zprvu nainicializovány náhodně:

```
initial_Theta1 = inicializaceVah(input_layer_size, hidden_layer_size);
initial_Theta2 = inicializaceVah(hidden_layer_size, num_labels);
```

výstupem každé iterace trénování sítě je hodnota účelové funkce, která se postupně zmenšuje.

```
Iteration     1 | Cost: 3.312223e+000
    Iteration     2 | Cost: 3.248407e+000
                    .
                    .
                    .
Iteration  1499 | Cost: 3.058311e-001
    Iteration  1500 | Cost: 3.058243e-001
```

Jakmile je neuronová síť naučena, je schopna začít číslice rozpoznávat. Rozpoznávání číslic probíhá tak, že neuronové síti je předložena sada vstupních dat. A ta na jejich základě vyprodukuje sadu výstupních, z nichž každá tato hodnota udává pravděpodobnost, že číslice patří do dané skupiny. Tedy vektor o 10 prvcích, například (0.0004, 0.0092, 0.0000, 0.0040, 0.0006, 0.9979, 0.0000, 0.0006, 0,0032, 0.0007), říká že výsledná číslice s 99,79 procent odpovídá skupině šestek a na druhou stranu z 0 procent odpovídá trojkám, či sedmičkám. Z tohoto vektoru je následně možné pomocí maxima určit, které skupině je číslice nejvíce podobná. S touto znalostí je tedy možné přikročit k samotnému testování na datech od Coursery:

Obrázek: 15 Správná predikce neuronové sítě: 0

Obrázek: 16 Správná predikce neuronové sítě: 7

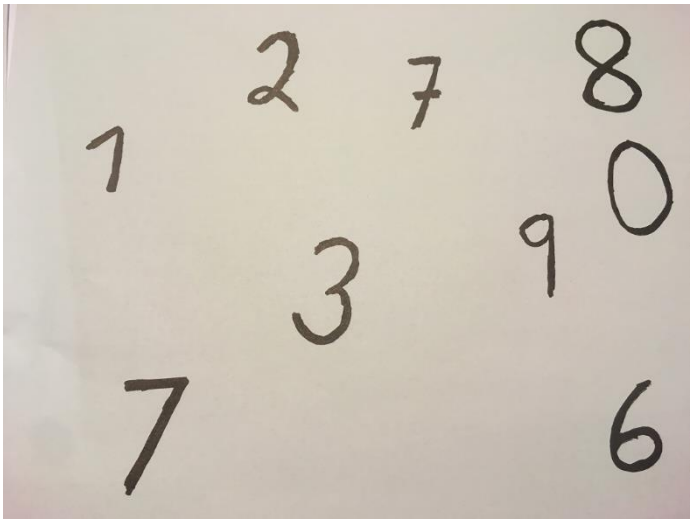
Obrázek: 17 Správná predikce neuronové sítě: 5

Necháme-li trénovacímu algoritmu na natrénování 1 500 iteracím, dosahuje takto naučená neuronová síť na normalizovaných datech úspěšnosti blízké 100 %. Při testování na 50 vzorcích dochází k maximální chybovosti 1 vzorku. Doba potřebná pro doběhnutí 1 500 iterací je 1 minuta a 58 vteřin. Pokud algoritmu omezíme počet iterací na 1/3, tedy na 500, pak se úspěšnost sítě pohybuje okolo 98 % a doba potřebná pro natrénování je 44 vteřin. Při 50 iteracích je pravděpodobnost úspěchu při testování stejná jako u 500 iterací.

Nejvíce problémovými číslicemi jsou špatně napsané 2 a 7. Problém může také nastat u nedotažené číslice 0, kterou neuronová síť odhaduje jako 6.

Druhou výzvou, která se pojí s tímto tématem je předzpracování obrázku s číslicemi do formátu, kterému je neuronová síť schopna porozumět. Tedy nejdříve proces segmentace, který obraz rozdělí na obrazy dílčí, z nichž každý bude obsahovat právě jednu číslici. A následně takto získaná data normalizovat na tvar a škálu hodnot, která je co možná nejpodobnější datům, na kterých byla neuronová síť naučena.

Algoritmu je poskytnut obrázek s naskenovanými číslicemi Obrázek: 18.



Obrázek: 18 Scan číslic pro testování sítě

Z tohoto obrazu jsou za pomoci sady transformací:

```
I = imread('lot.png');
imshow(I);
G = grayscale(I);
imshow(G);
G = medfilt2(G);
T = otsu(G);
G = hranice(G);
G= prahovani(G,T);
G=imfill(G, 'holes');
strel=[1,1,1,1,1;1,1,1,1,1;1,1,1,1,1;1,1,1,1,1;1,1,1,1,1];
G= pridani(G,strel);
[G sum] = spojovani(G,[1,1,1;1,1,1;1,1,1]);
stats = regionprops(G,'BoundingBox');
rectangles = zeros(10,4);
for i=1:size(stats,1)
rectangles(i,:) = stats(i).BoundingBox;
end
```

získány obrázky číslic, které jsou následně předloženy neuronové síti pro rozpoznání. Výsledek je viditelný na obrázcích

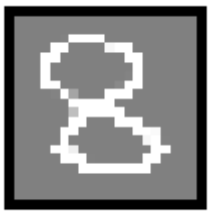
Obrázek: 19 Předzpracovaná číslice 1, špatně rozpoznaná jako 7

Obrázek: 20 Předzpracovaná číslice 7, špatně rozpoznaná jako 2

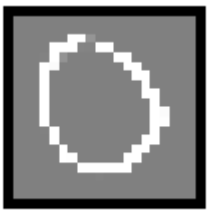
Obrázek: 21 Předzpracovaná číslice 2, správně rozpoznaná

Obrázek: 23 Předzpracovaná číslice 9, špatně rozpoznaná jako 7

Obrázek: 25 Předzpracovaná číslice 3, správně rozpoznaná 3



Obrázek: 24 Předzpracovaná číslice 8, špatně rozpoznaná jako 3



Obrázek: 26 Předzpracovaná číslice 0, správně rozpoznaná

Obrázek: 27 Předzpracovaná číslice 6,
správně rozpoznaná

Z výsledků je patrné, že ke správně fungující neuronové síti je zapotřebí ji natrénovat na datech, která mají stejné charakteristiky jako následně rozpoznávané číslice. Tedy na datech, na kterých proběhla stejná sada funkcí pro předzpracování jako na rozpoznávaných číslicích. Neboť pakliže data odpovídají tvarem, ale hodnotami se liší, tato diference následně způsobí špatnou předpověď neuronové sítě v zhruba 55 % případů.

5 Závěr

V této práci byly nejdříve rozebrány metody používané při předzpracování obrazu, které jsou nezbytné pro extrakci obrazové informace. Ta je následně možná matematicky popsat za pomoci rysů, které jsou předány algoritmu neuronových sítí, který je na jejich základě schopen informaci rozpoznat.

Rozpoznávání rysů na neuronové síti předchází proces jejího učením, který byl v práci popsán taktéž. Nejprve na příkladu algoritmu lineární regrese a později složitější logistické regrese, která je využívána právě v neuronových sítích. Následně byl rozebrán problém rozpoznávání číslic a jeho definice za pomoci algoritmu jeden proti všem. Na základě tohoto rozboru bylo následně možné definovat strukturu neuronové sítě. Ta má tvar 40 x 25x 10. V neposlední řadě byl vysvětlen učící algoritmus neuronové sítě zvaný zpětná propagace. Ten byl posledním nezbytným teoretickým prvkem pro praktickou implementaci problému.

Pro implementaci neuronové sítě byl použit programovací jazyk Matlab a byly zmíněny důležité použité funkce – a následně byl představen zdrojový kód aplikace. Tato implementace neuronové sítě dosáhla úspěšnosti rozpoznávání okolo 95 % na datech poskytnutých online kurzem Coursera. Ve studii by šlo pokračovat například experimentováním s jiným počtem neuronů na skryté vrstvě, případně aplikování stejného algoritmu pro rozpoznávání jiných objektů než pouze číslic. K tomuto by stačilo pouze sehnat relevantní datovou sadu pro učení a následné testování neuronové sítě.

6 Seznam použité literatury

[1] ANDREOPOULOS, Alexander a John K. TSOTSOS. 50 Years of object recognition: Directions forward. *Computer Vision and Image Understanding* [online]. 2013, **117**(8), 827-891 [cit. 2016-02-05]. DOI: 10.1016/j.cviu.2013.04.005. ISSN 10773142. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S107731421300091X

[2] Three algorithms for converting color to grayscale. John D. Cook [online]. 2009 [cit. 2016-02-05]. Dostupné z: http://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/

[3] Milan Šonka, Václav Hlaváč, Roger Boyle : Image Processing Analysis, and Machine Vision, International student edition 2008, Third edition

[4] Andrew Ng. Machine Learning. Lecture 1 – 5 [video]. Stanford University [online]. Coursera, 2015. [Vid. 17.10.2011]. Dostupné z: https://www.coursera.org/learn/machine-learning/

[5] MUNAKATA, Toshinori. Fundamentals of the new artificial intelligence: neural, evolutionary, fuzzy and more. 2nd ed. London: Springer, 2008, xi, 255 p. ISBN 9781846288395.

[6] SAMARASINGHE, Sandhya. Neural networks for applied sciences and engineering: from fundamentals to complex pattern recognition. Boca Raton, FL: Auerbach, c2007. ISBN 9780849333750.

[7] THEODORIDIS, Sergios. Introduction to pattern recognition: a MATLAB approach. Amsterdam: Academic Press, 2010. ISBN 978-0-12-374486-9.

[8] Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

[9] James R. Parker. Algorithms for Image Processing and Computer Vision. John Wiley and Sons, 1997.

[10] SOLOMON, Chris a Toby BRECKON. *Fundamentals of digital image processing a practical approach with examples in Matlab*. Chichester: Wiley-Blackwell, 2011. ISBN 9780470689776.

[11] Mitchell, T. (1997). Machine Learning. McGraw Hill. p. 2. ISBN 0-07-042807-7

[12] Článek. docs.opencv.org [online]. Poslední změna 30. června 2017. Dostupné z: http://docs.opencv.org/2.4/modules/ml/doc/neural_networks.html

7 Seznam obrázků

8 Přílohy

Přílohy se nalézají na přiloženém CD. Toto medium obsahuje všechny funkce a data potřebná ke spuštění funkcí run_neural_only a run_with_segmentation.

Seznam funkcí:

1. run_neural_only
2. run_with_segmentation
3. data.mat
4. costFunkce.m
5. fmincg.m[4]
6. grayscale.m
7. hadani.m
8. hranice.m
9. inicializaceVah.m
10. cisla.png
11. otsu.m
12. otsuCalculation.m
13. prahovani.m
14. pridani.m
15. sigmoidFunkce.m
16. sigmoidFunjceDerivace.m

17. spojovani.m

18. toGray.m[4]

19. toGrayOriginal.m[4]

20. zobrazeniDat.m

21. 4.jpg

22. 6.jpg

23. 7.jpg

24. 8.jpg

25. 11.jpg

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2015/2016

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Kratochvíl Pavel	Vrcha 62, Sobětuchy - Vrcha	I1300995

TÉMA ČESKY:

Detekce objektů v obraze

TÉMA ANGLICKY:

Shape recognition

VEDOUCÍ PRÁCE:

Ing. Ondřej Klapka - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl
1. Metody detekce a klasifikace objektů v obraze
2. Možnosti umělých neuronových sítí a možnosti jejich aplikace na řešený problém
3. Navrhnout řešení, využívající některý typ klasifikátoru založený na neuronových sítích
4. Implementace navrženého řešení
5. Zhodnotit dosažené výsledky

SEZNAM DOPORUČENÉ LITERATURY:

1.Sergios Theodoridis, Konstantinos Koutroumbas (2003): Pattern recognition
2.David A. Forsyth, Jean Ponce (2012): Computer Vision: A Modern Approach
3.Toshinoru Munakata (2008): Fundamentals of the New Artificial Intelligence
4.Sandhya Samarasinghe (2006): Neural Networkd for Applied Sciences and Engineering

Podpis studenta: ..... **Datum:** .....

Podpis vedoucího práce: ..... **Datum:** .....